Projeto e Análise de Algoritmos
Exercícios: **Análise Assintótica, Programação Dinâmica e Memoização**

1. **Seja** $P : N \to N$ **uma função definida da seguinte forma:** $P(0) = P(1) = P(2) = P(3) = P(4) = 0$ **e, para** $n \geq 5$**,**

$$P(n) = P(\lfloor \tfrac{n}{2} \rfloor) + P(\lfloor \tfrac{n}{2} \rfloor + 1) + P(\lfloor \tfrac{n}{2} \rfloor + 2) + n.$$

   (a) Escreva um algoritmo recursivo puro que recebe um número n como entrada e retorna o valor exato de $P(n)$. Calcule a complexidade do seu algoritmo.

Algoritmo 1: P(n)

```
1 início
2 |  if n ≤ 4 then
3 |  |  retorne 0;
4 |  else
5 |  |  retorne P(⌊n/2⌋) + P(⌊n/2⌋ + 1) + P(⌊n/2⌋ + 2) + n;
6 |  end
7 fim
```

Complexidadde:

A recurssão gera a seguinte equação de recorrência:

$$T(n) = T(\lfloor \tfrac{n}{2} \rfloor) + T(\lfloor \tfrac{n}{2} \rfloor + 1) + T(\lfloor \tfrac{n}{2} \rfloor + 2) + n$$

Desprezando as constantes, temos:

$$T(n) = 3T(\tfrac{n}{2}) + \Theta(1)$$

A partir do Teorema Mestre ($T(n) = aT(\frac{n}{b}) + c$), podemos tirar:

$$a = 3; b = 2; c = 0$$

Logo, a complexidade é:

$$\Theta(n^{\log_2 3})$$

(b) Escreva um algoritmo de programação dinâmica para o mesmo problema e calcule sua complexidade.

i. Primeiro precisamos verificar se uma parte da solução ótima é solução ótima para uma parte do problema:
OK.

ii. Em seguida, criar a recursão:

$$P(n) = \begin{cases} 0, & \text{se } k < 5 \\ P(\lfloor \frac{n}{2} \rfloor) + P(\lfloor \frac{n}{2} \rfloor + 1) + P(\lfloor \frac{n}{2} \rfloor + 2) + n, & \text{se } n \geq 5 \end{cases}$$

iii. E então, criar o algoritmo:

Algoritmo 2: Dinamico(n)

```
1 início
2 |  criar vetor P[n];
3 |  P[0] ← P[1] ← P[2] ← P[3] ← P[4] ← 0;
4 |  i ← 5;
5 |  repita
6 |  |  P[i] ← P[⌊i/2⌋] + P[⌊i/2⌋ + 1] + P[⌊i/2⌋ + 2] + i;
7 |  até i = n;
8 fim
```

Complexidade: Por conta do *for* (linhas 5 à 7), a complexidade do algoritmo é $\Theta(n)$.

(c) Escreva um algoritmo de memoização para o mesmo problema e calcule sua complexidade.

Algoritmo 3: Memo(n)

```
1  início
2  |  criar vetor P[n];
3  |  i ← n;
4  |  P[n] ← −1;
5  |  while i/2 > 0 do
6  |  |  P[⌊i/2⌋] ← +P[⌊i/2⌋+1]+P[⌊i/2⌋+2]+P[⌊i/2⌋+3]+P[⌊i/2⌋+4] ← −1;
7  |  |  i ← i/2;
8  |  end
9  |  retorne MemoRec(P, n);
10 fim
```

Complexidade: Como o iterador do *while* é sempre dividido pela metade, ele se torna menor ainda a cada nível, assim, a complexidade por ser dada por $\Theta(\log_2 n)$.

Algoritmo 4: MemoRec(P,n)

```
1  início
2  |  if P[n] ≠ −1 then
3  |  |  retorne P[n];
4  |  else
5  |  |  P[n] ←
   |  |    MemoRec(⌊n/2⌋)+MemoRec(⌊n/2⌋+1)+MemoRec(⌊n/2⌋+2)+n;
6  |  |  retorne P[n];
7  |  end
8  fim
```

Complexidade: Como o n é sempre dividido pela metade a cada recursão, a complexidade por ser dada por $\Theta(\log_2 n)$.

Complexidade Final: Portanto, a complexidade geral é $\Theta(\log n)$